

Container Orchestration and Management I

CNAE – Cloud Native Architecture & Engineering



Prof. Dr.-Ing. Stefan Tai
Sebastian Werner, Elias Grünewald, Nicola Leschke, Maria Borges

Information Systems Engineering
TU Berlin

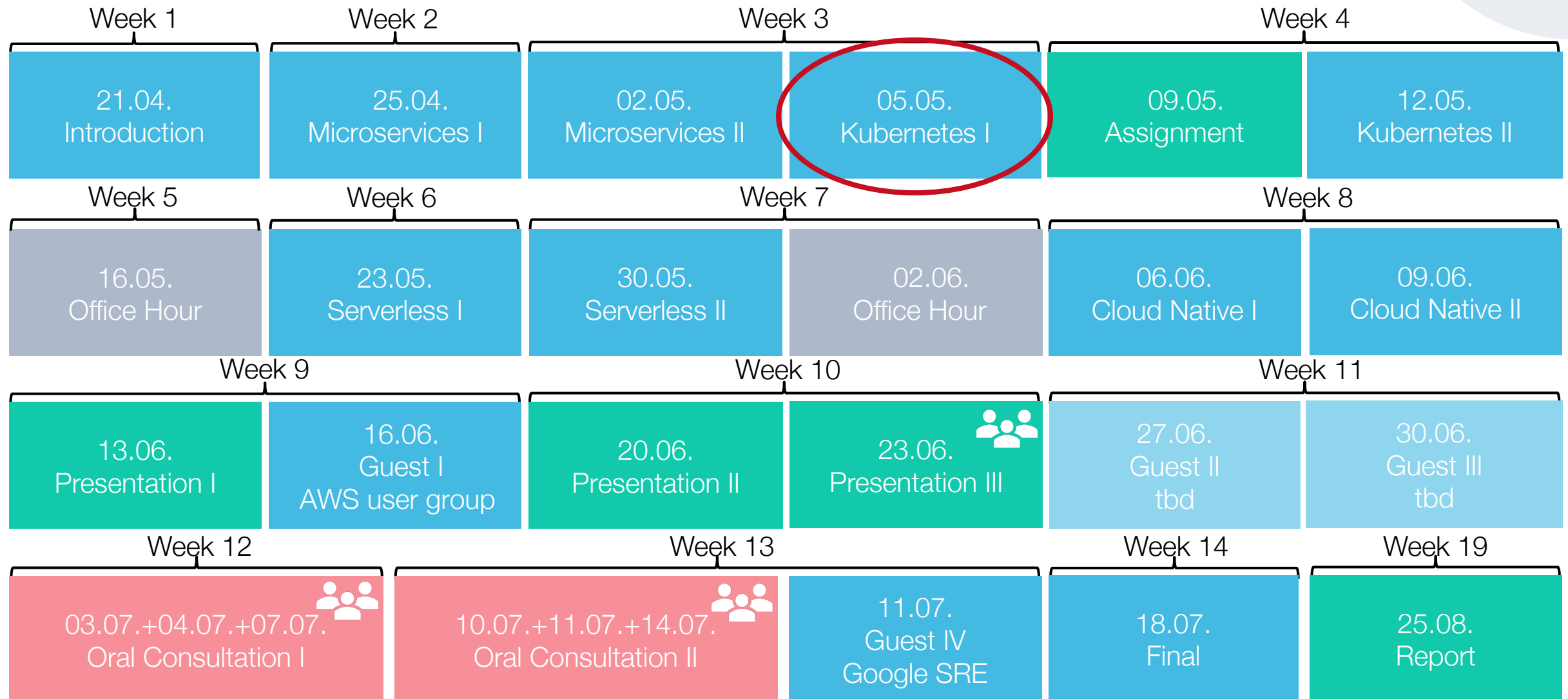
Formal registration deadline: **15.05.2023, 23:59**

Registration via Moses:

<https://moseskonto.tu-berlin.de/moses/modultransfersystem/index.html>

- select "MTS / Module exams registration/deregistration" after logging in
- Select right course and semester!
- instructions with screenshots are only available in [German](#)

Schedule (still preliminary)



Recap: Microservices I + II

Microservices...

...are **autonomously deployable services** providing a **focused amount** of functionality.

...are **standardized with respect to interfaces and deployment** procedures, but not things „below the surface“.

...require **organizational structure** and a **DevOps** culture.

...require conscious **design decisions** w.r.t., e.g., data management, API design and communication.

...can be engineered to meet **different qualities**.

Agenda



Container orchestration and management I

Containerization, Container orchestration, Case studies, Challenges

Container orchestration and management II

Infrastructure as Code, Release strategies, Service Discovery

Mini example



```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def helloworld():
    return "Hello World!"

@app.route("/get_data")
def getdata():
    return "Your data"

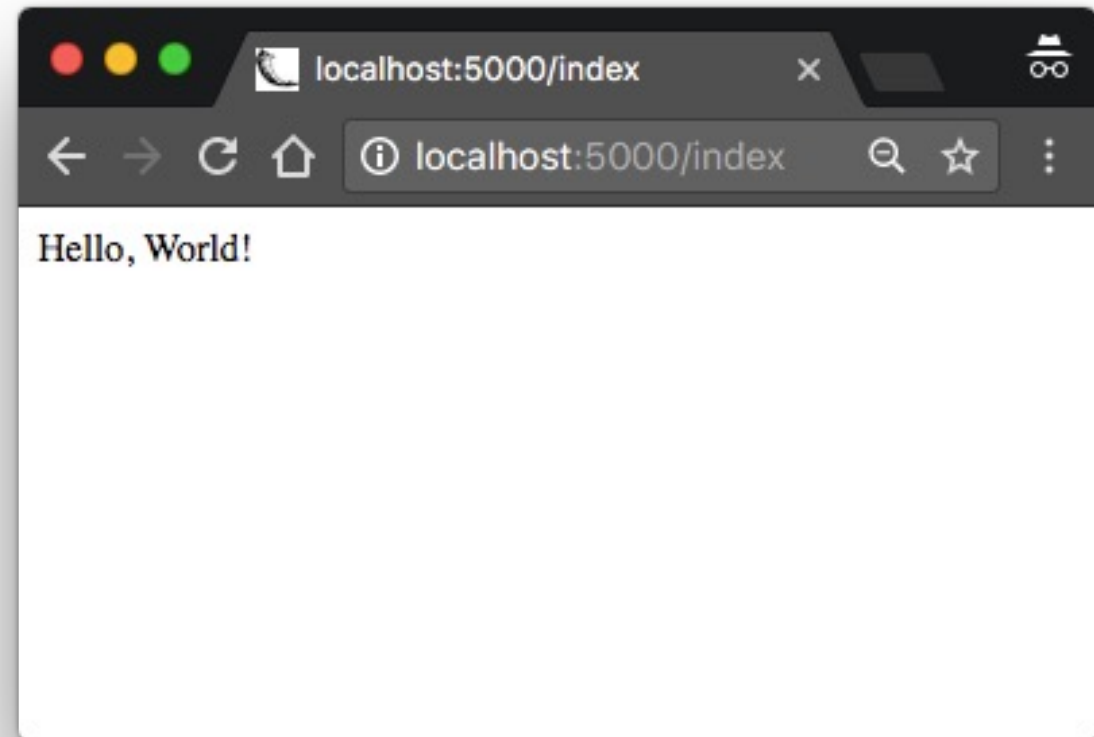
if __name__ == "__main__":
    app.run()
```

Mini example

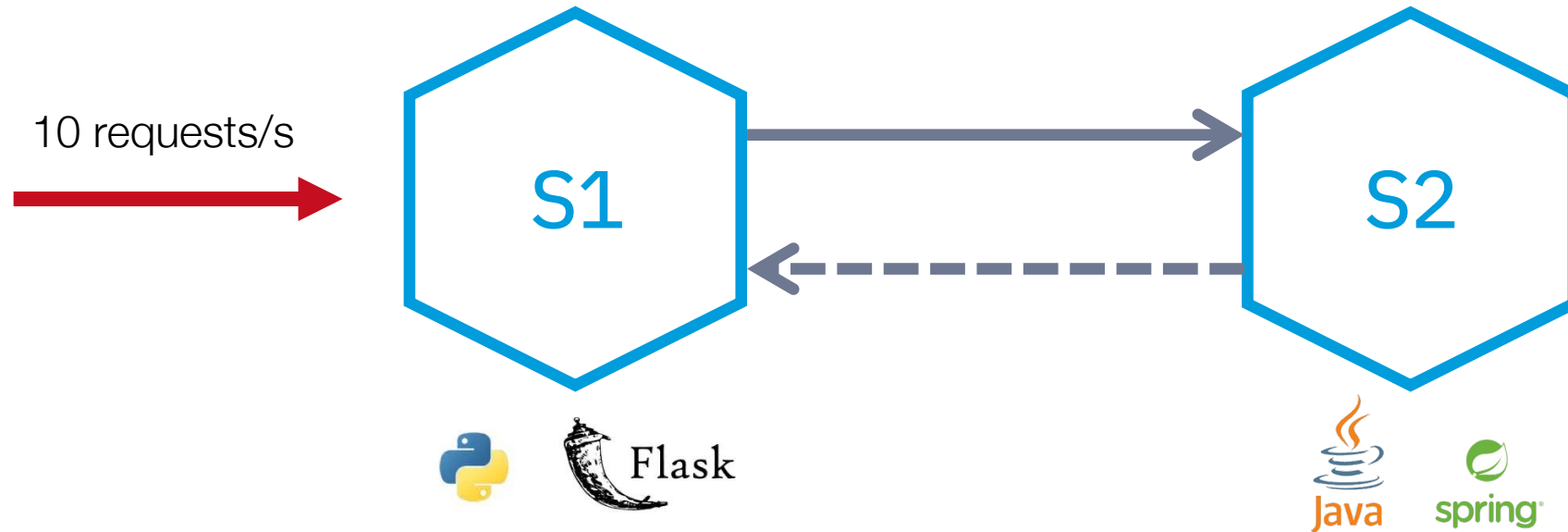


```
$ sudo apt install python3  
$ sudo apt install python3-pip  
  
$ pip install flask  
  
$ export FLASK_APP=app.py  
  
$ flask run
```

Mini example



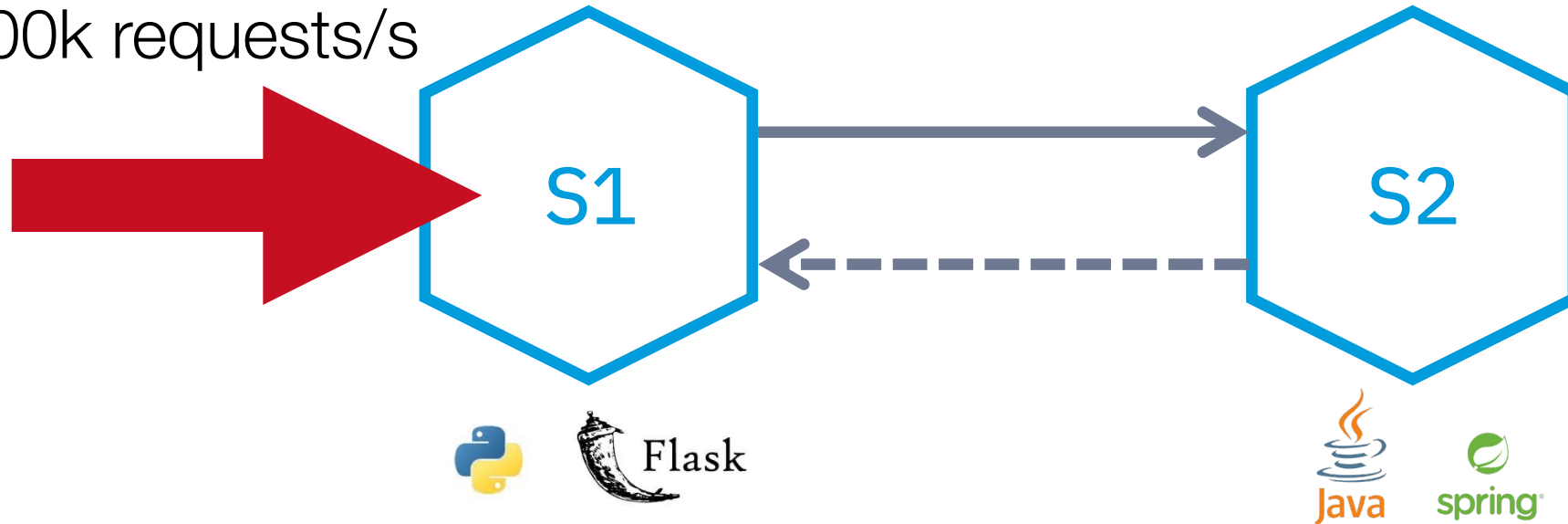
Mini example



See, e.g., <https://spring.io/guides/gs/spring-boot/>

Mini example

100k requests/s

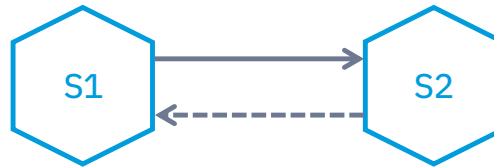


Service coupling

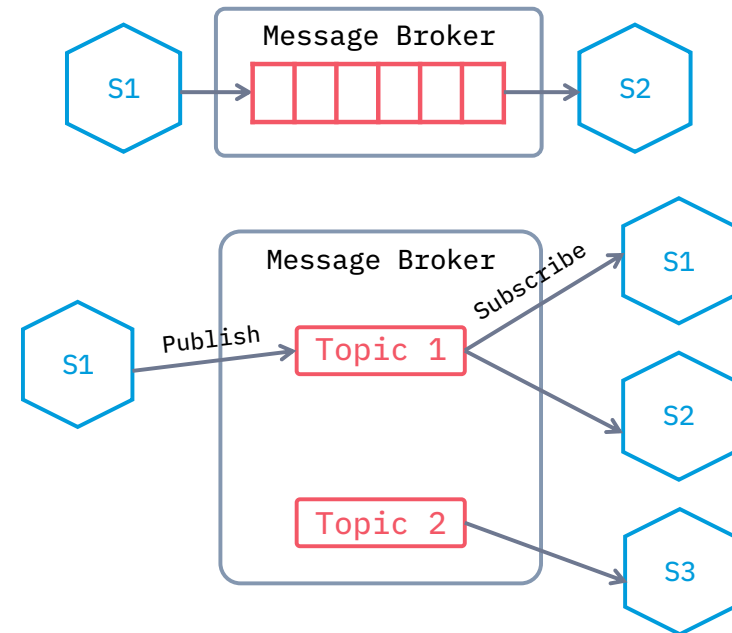
synchronous



asynchronous



message-oriented



tight coupling

loose coupling

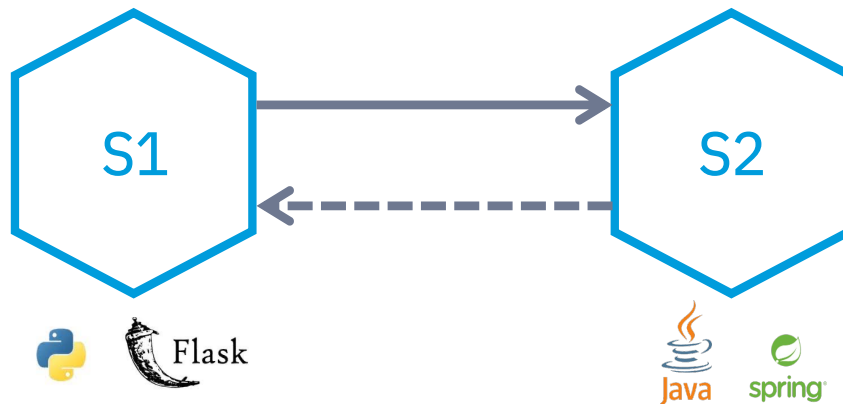
Further challenges

Scalability (e.g., too many requests for a single machine)

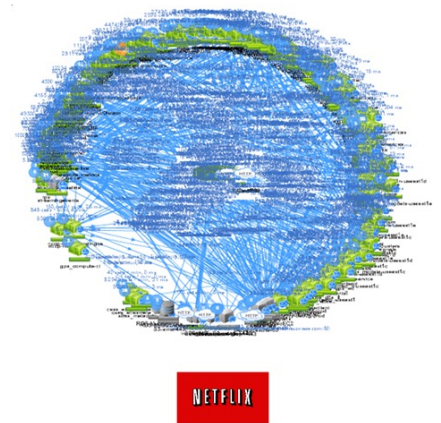
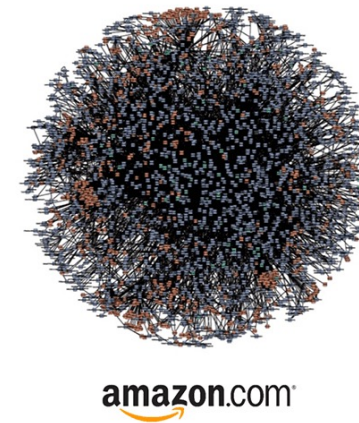
DevOps, distributed development and deployment (i.e., running on any machine)

Security (e.g., remote code execution)

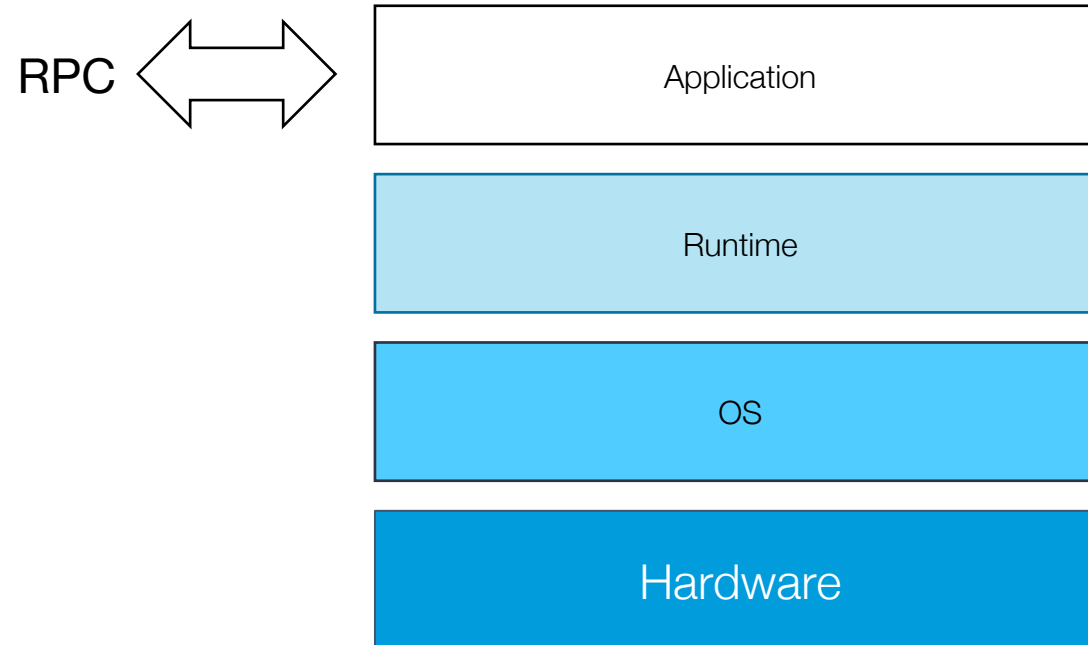
...



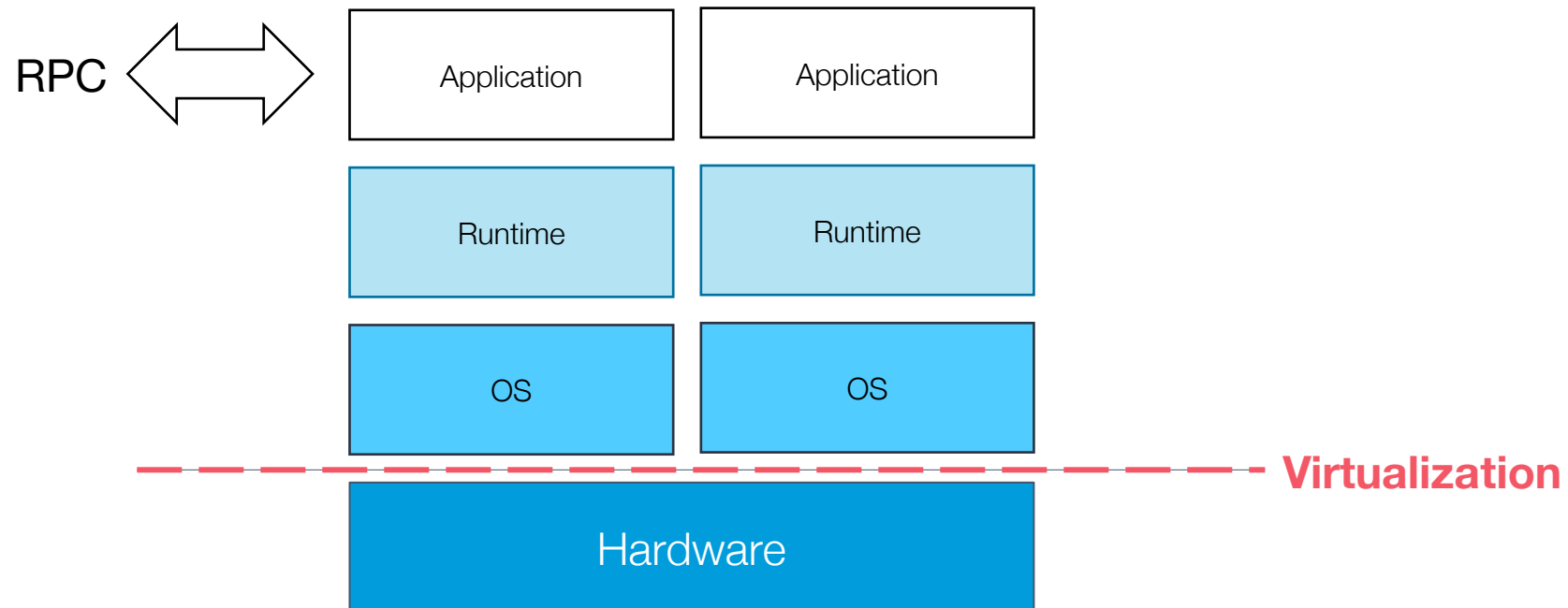
vs.



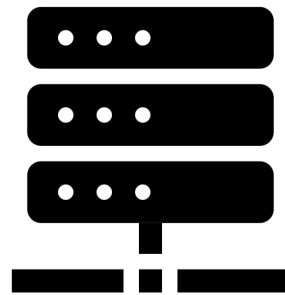
Traditional Deployment Model



1st Generation - Virtual Machines



Motivation: Containerization



VM 1

```
$ sudo apt install python3
$ sudo apt install python3-pip

$ pip install flask

$ export FLASK_APP=app.py

$ flask run
```



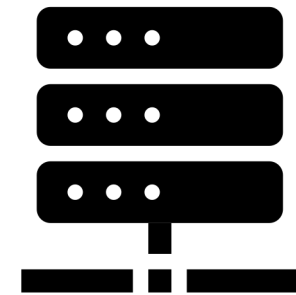
VM 2

```
$ sudo apt install python3
$ sudo apt install python3-pip

$ pip install flask

$ export FLASK_APP=app.py

$ flask run
```



VM 3

```
$ sudo apt install python3
$ sudo apt install python3-pip

$ pip install flask

$ export FLASK_APP=app.py

$ flask run
```

Virtual machines

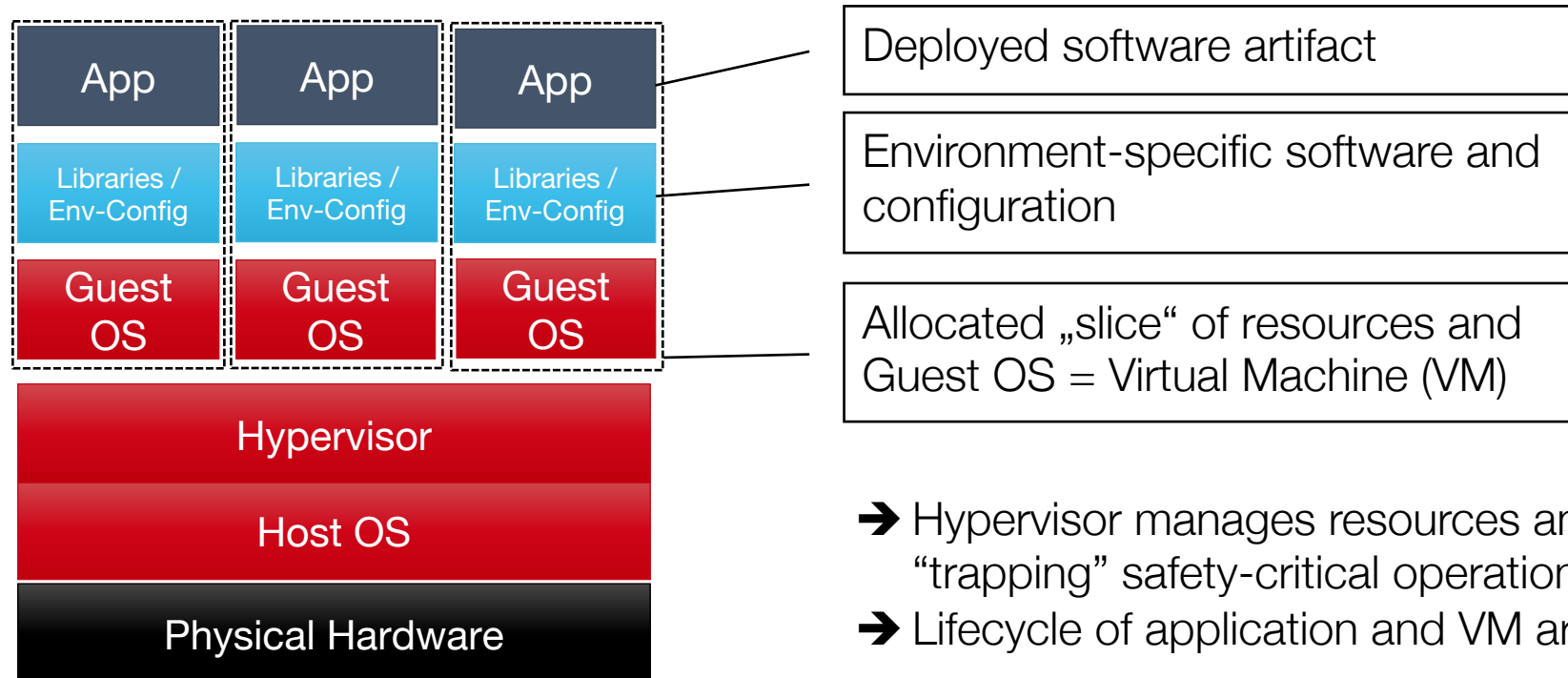
- Install and manage: OS, software, your application and Load Balancing
- Provisioning: takes minutes
- Runtime for: several processes or applications
- Cost: per time and resources

Containers

- **Layered image architecture**
- **Install and manage: software, your application and load balancing**
- **Provisioning: takes seconds**
- **Intended for a single application/process**
- **Cost: per time and resources**

Virtualization

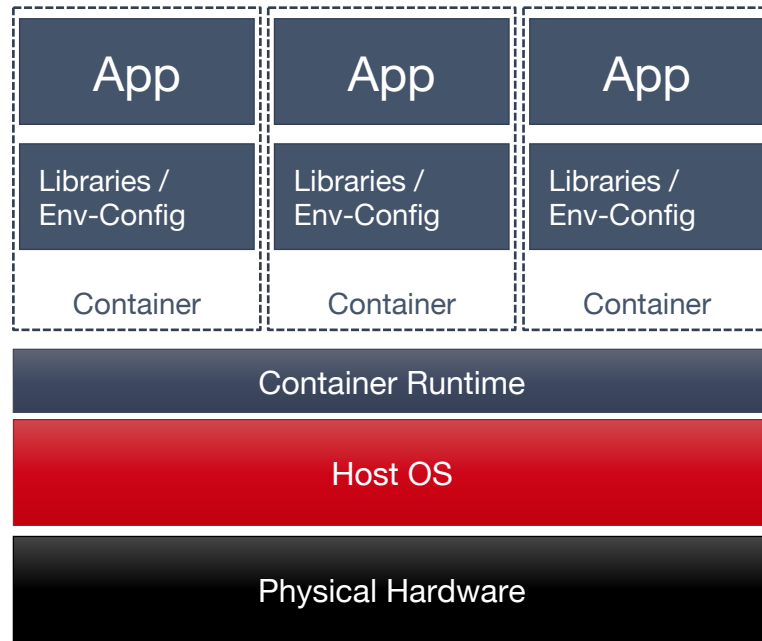
Hypervisor-based Virtualization



- ➔ Hypervisor manages resources and ensures isolation by “trapping” safety-critical operations (e.g., memory allocation)
- ➔ Lifecycle of application and VM are decoupled

Virtualization (contd.)

Operating System-based Virtualization (aka “Container Virtualization”)

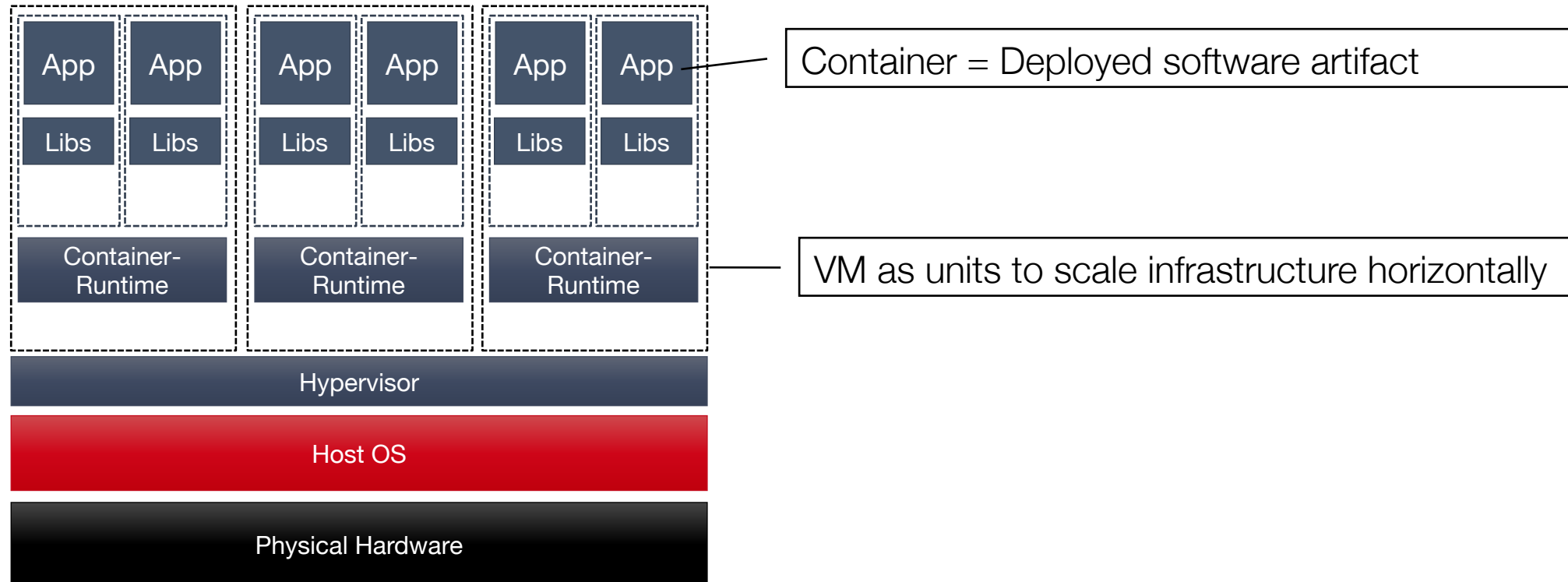


Deployed software artifact

- ➔ Container runtime manages resources
- ➔ All containers share the same OS kernel
- ➔ OS-level mechanisms (user management, root-level hardening) are used to ensure isolation
- ➔ Lifecycle of application and container are coupled (a container is a regular process running in its own “namespace”)

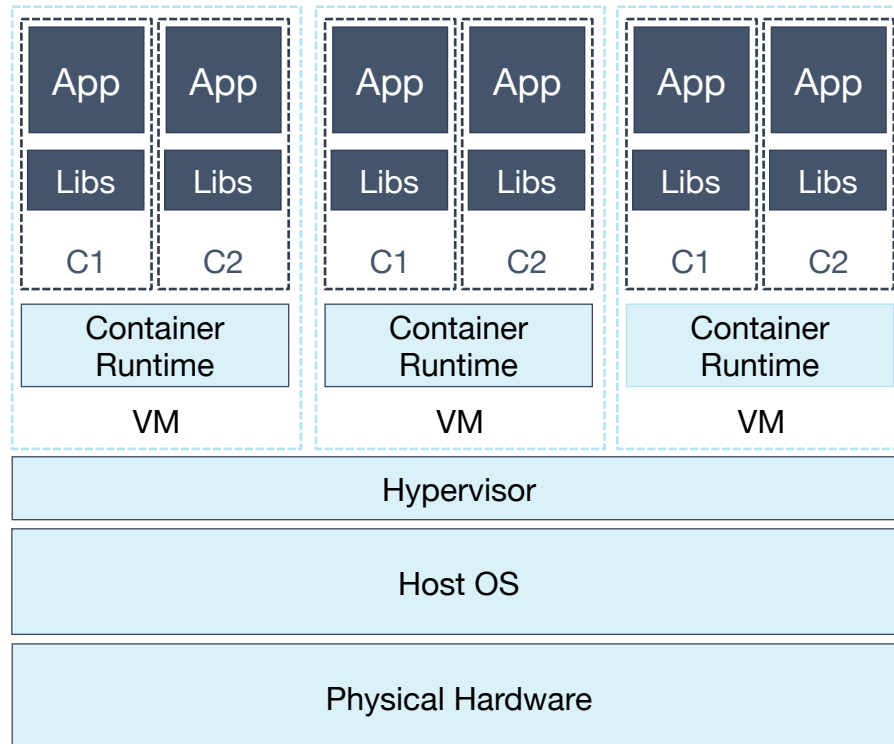
Practical Virtualization in the Cloud

Application-level Virtualization and Resource-level Virtualization



Practical Virtualization in the Cloud, e.g., PaaS

Different parties might be interested in different trade-offs regarding isolation etc.

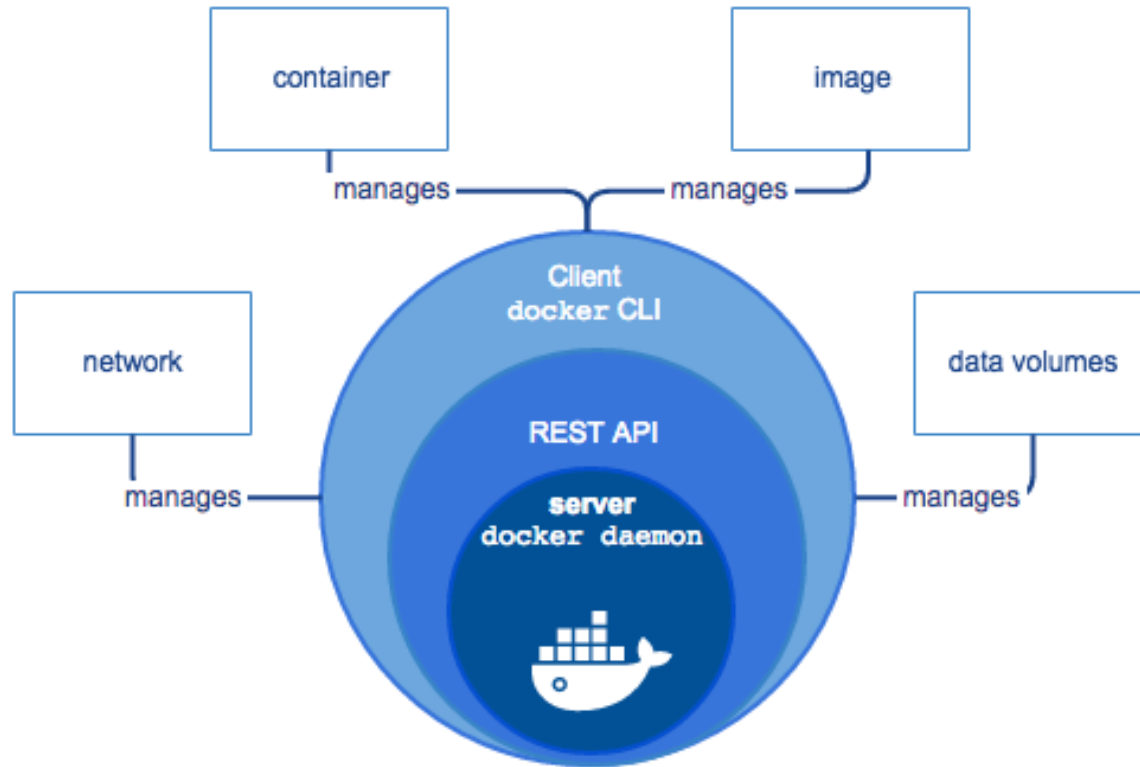


Managed by user

Managed by provider

- ➔ OS-level virtualization has lower overhead
- ➔ Hypervisors provide a second layer of security
- ➔ In practice, they are often combined

Containerization with Docker



Idea: OS-level virtualization

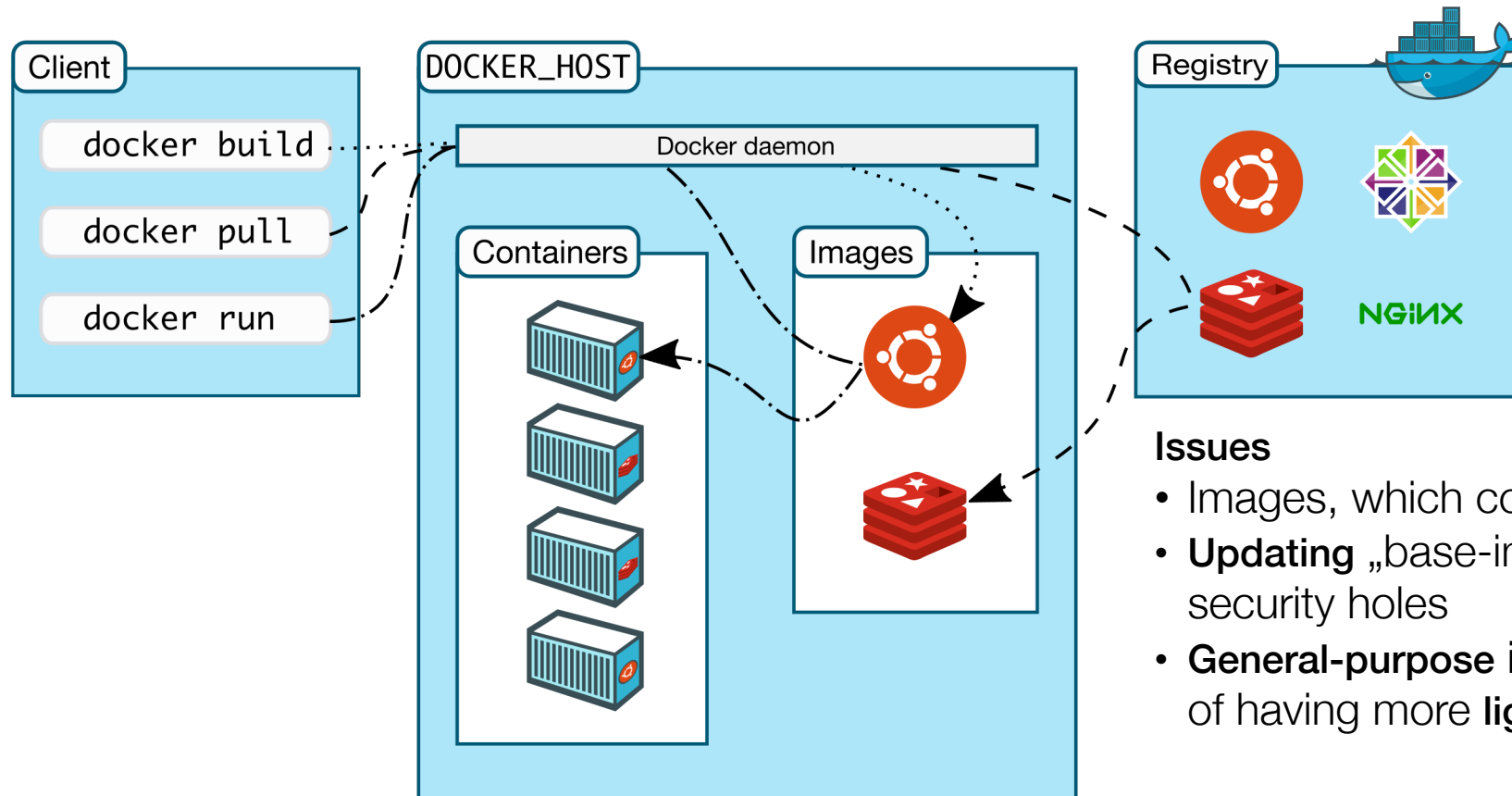
- All containers share the same host
- Host OS manages isolation

Docker (or other container management) requires hosts to be equipped with a server component and an API

Docker

Docker manages **images** and **containers**

- Images are stored in **registries** and **layered**, that means they build upon each other incrementally; this reduces transferred data significantly if layers are re-used
- **Client and host are not necessarily the same**, the client can connect to remotely available API-hosts



Issues

- Images, which contain **untrusted** software
- **Updating** „base-images“ to avoid exploitable security holes
- **General-purpose** images nullify the advantages of having more **light-weight** deployment artifacts

Source: <https://docs.docker.com/engine/docker-overview>

Case study: Docker

```
# Use an official Python runtime as a parent image
FROM python:3.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at
/app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r
requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

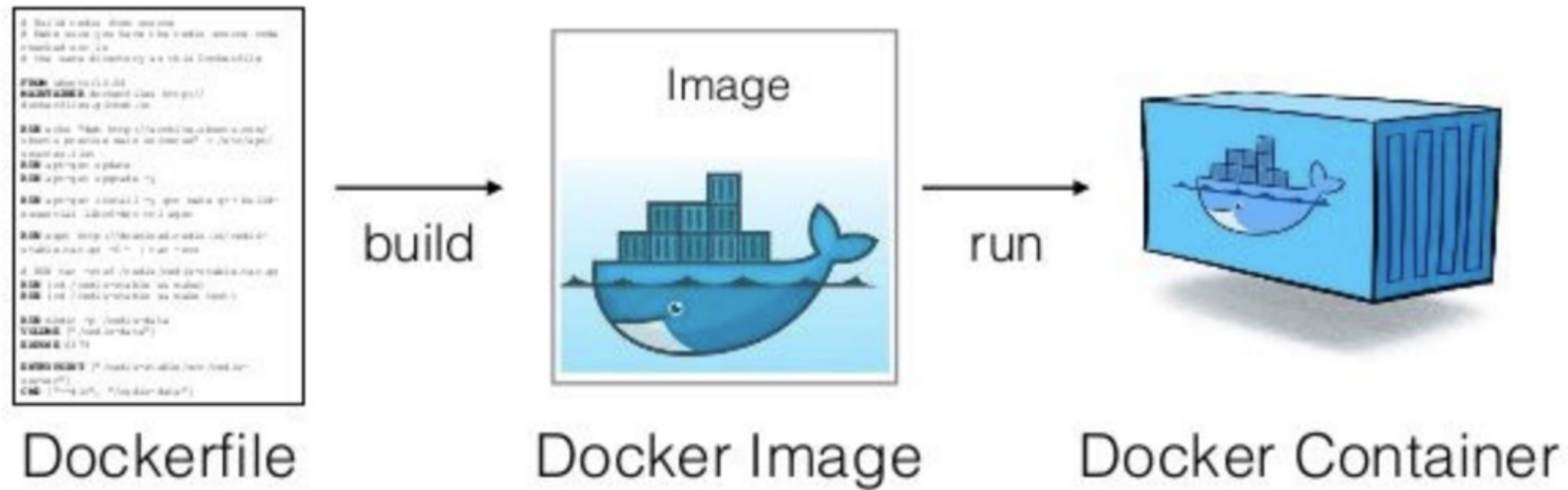
Source: <https://docs.docker.com/get-started/part2/#define-a-container-with-dockerfile>

Based on a **Dockerfile**,
a container image is built

The image is pushed to a registry
(i.e., repository for container images)

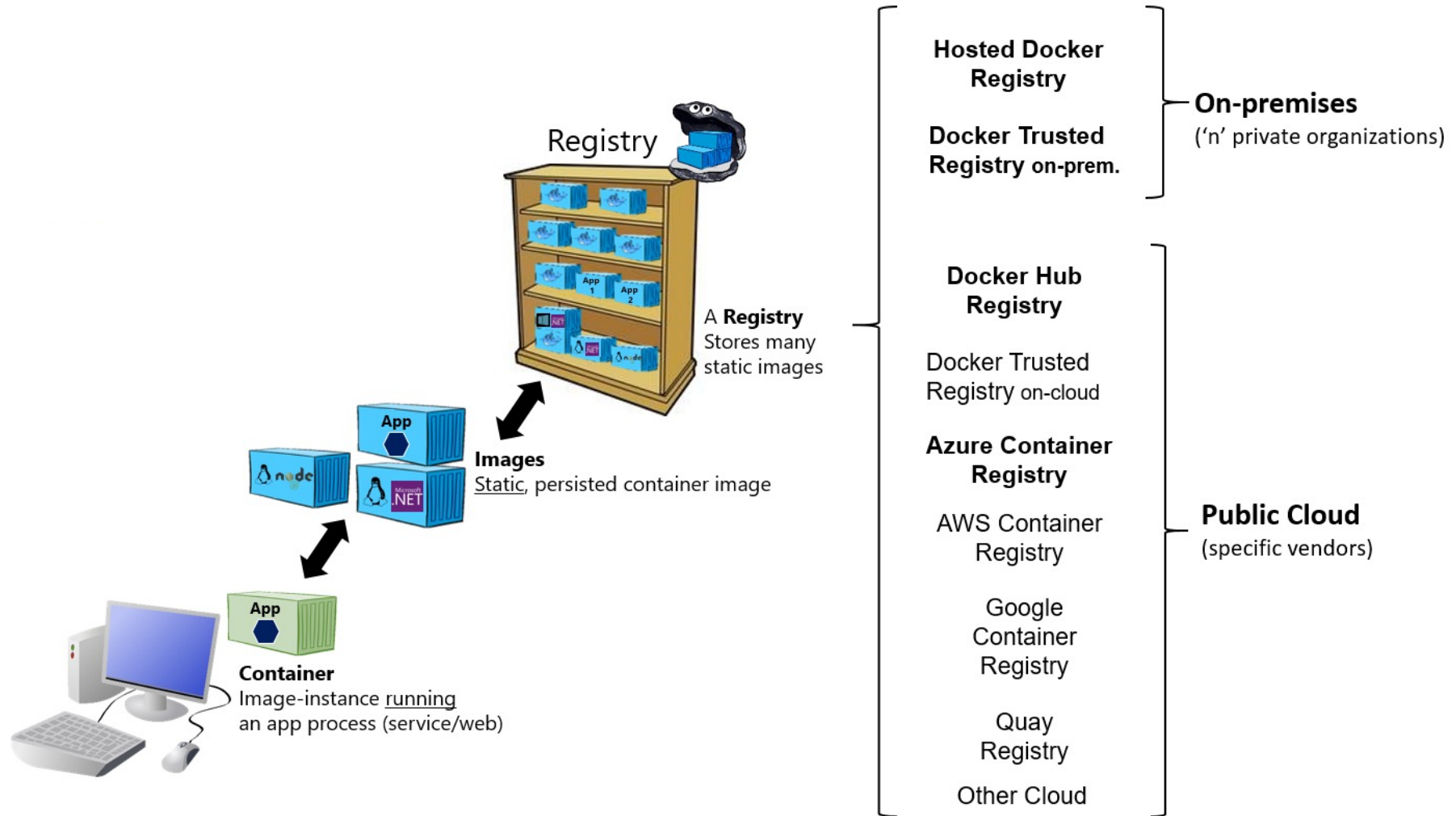
Controlled through CLI (scriptable)

Enabling Maintainability



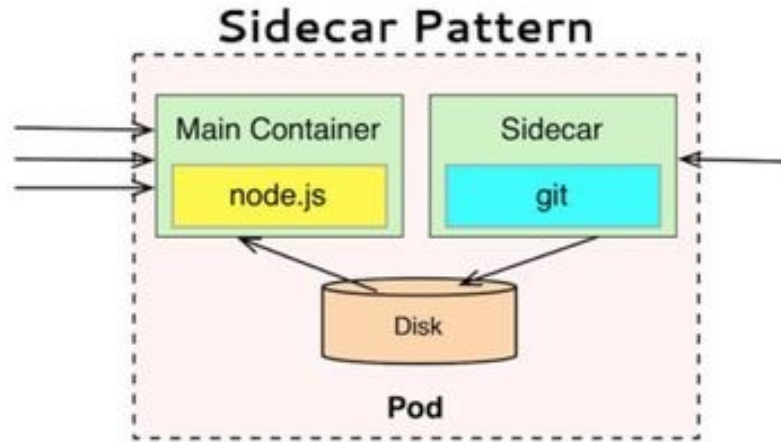
<https://cto.ai/blog/docker-image-vs-container-vs-dockerfile/>

Container registries

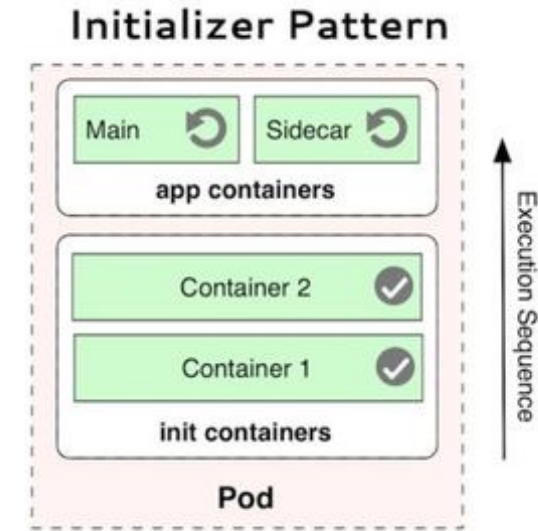


<https://learn.microsoft.com/de-de/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries>

Container patterns (examples)



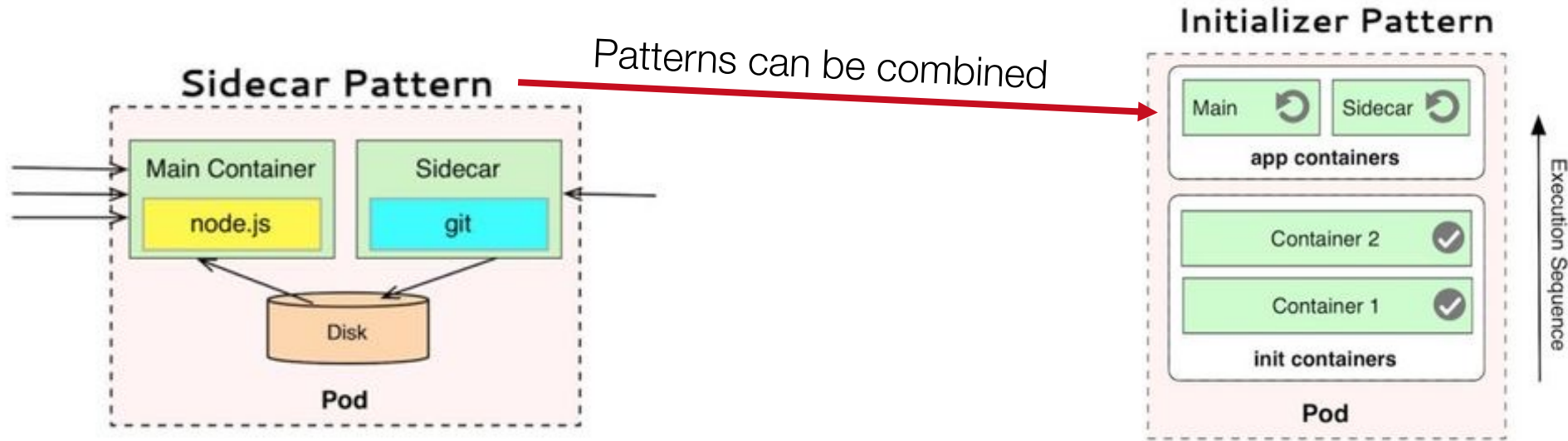
a sidecar container extends and enhances the functionality of a pre-existing container without changing it



init containers allow separation of initialization related tasks from the main application logic

Source: <https://www.infoq.com/articles/kubernetes-effect/>

Container patterns (examples)

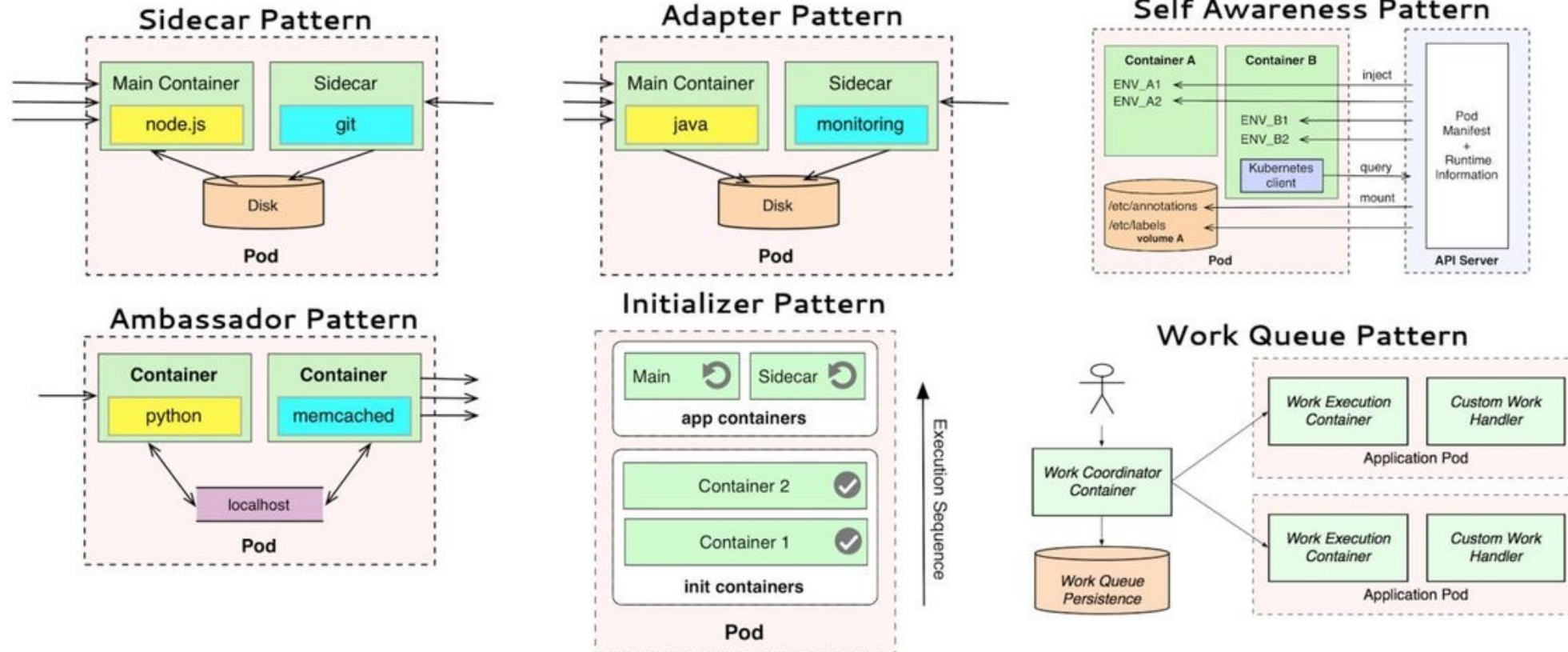


a sidecar container extends and enhances the functionality of a pre-existing container without changing it

init containers allow separation of initialization related tasks from the main application logic

Source: <https://www.infoq.com/articles/kubernetes-effect/>

Container patterns



Source: <https://www.infoq.com/articles/kubernetes-effect/>



Burns, Brendan, and David Oppenheimer. "Design patterns for container-based distributed systems." In 8th USENIX workshop on hot topics in cloud computing (HotCloud 16). 2016. https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf

Container platforms and the cloud

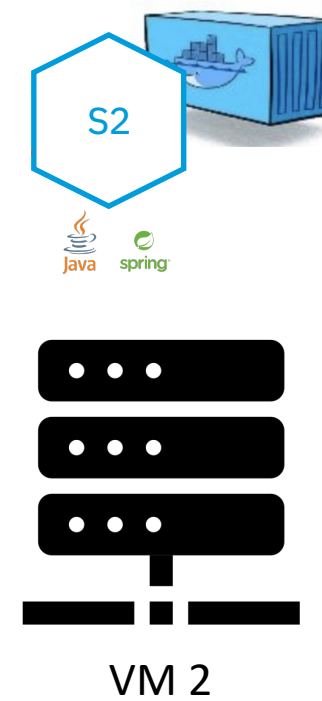
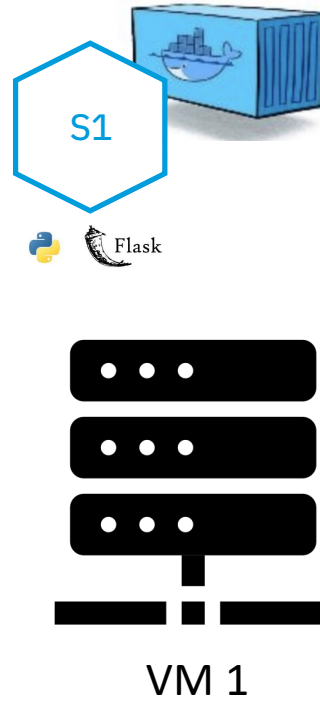
Cloud providers offer services which facilitate the deployment of containers

- Google Container Engine
- Amazon EC2 Container Service
- Microsoft Azure Container Service

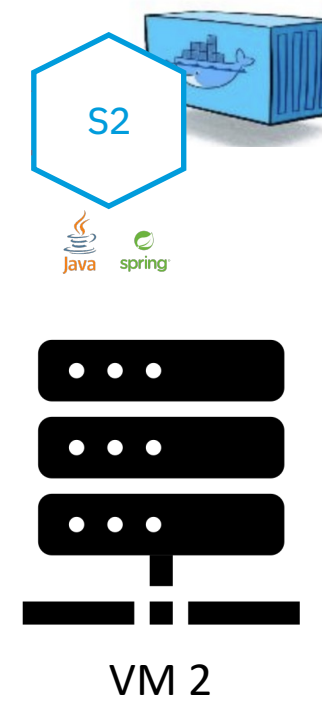
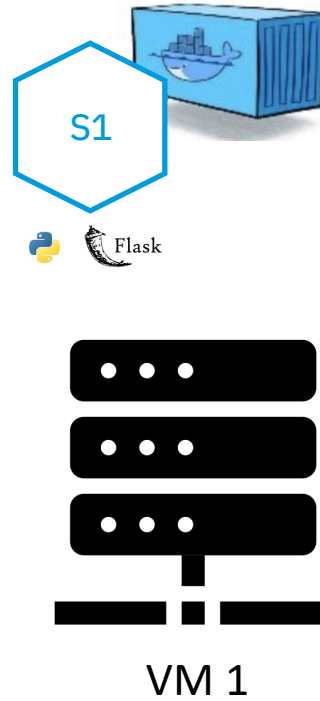
→ Containers are already part of the Cloud
→ New Cloud services make use of their characteristics

(e.g., „Serverless“ approaches, such as AWS Lambda or Google Cloud Functions)

Running containers , now what?



Running containers , now what?



Kubernetes

Production-grade container orchestration

De-facto industry standard

Google-grown (~2014), then donated to the Cloud Native Computing Foundation

For its evolution, see

<https://blog.risingstack.com/the-history-of-kubernetes/>

**Declarative infrastructure descriptions,
i.e., you describe the desired state,
Kubernetes tries to take of it**



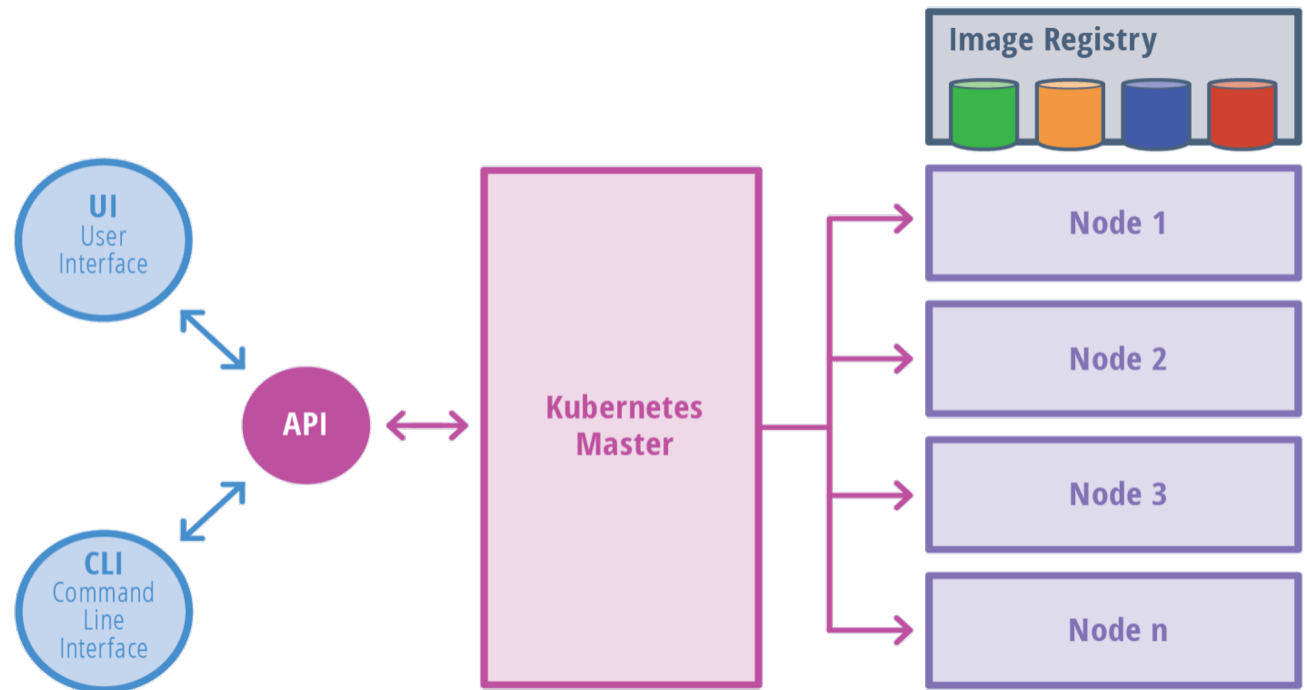
Practical Microservice Deployment – Kubernetes

Platforms enable managed, large-scale container deployments and orchestration

- Kubernetes (Google)
- Mesosphere Marathon (Apache)

Automate deployment, scaling and management of container-based deployments on a large scale

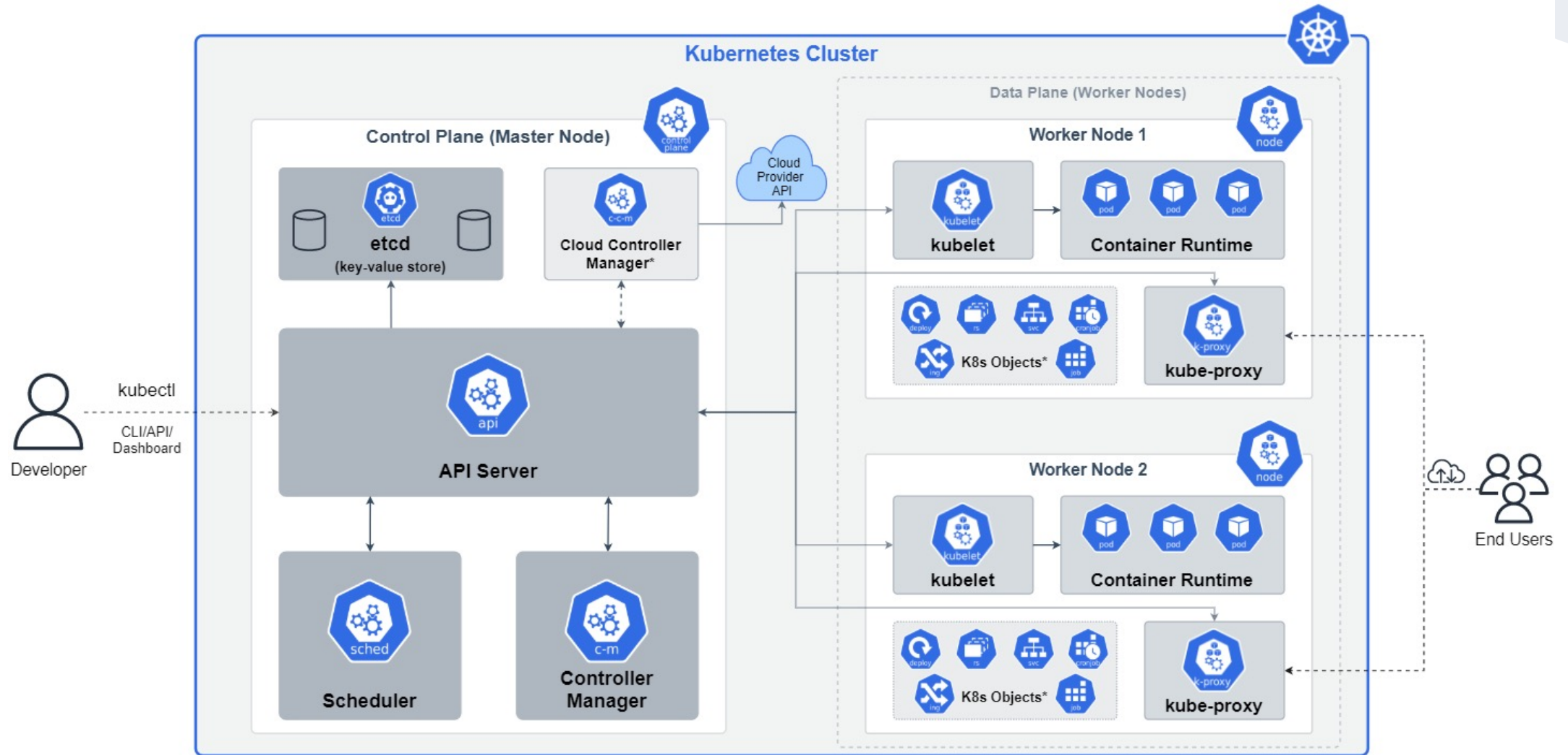
Describe an application using Kubernetes-specific, so-called, controllers



Source: Janakiram MSV

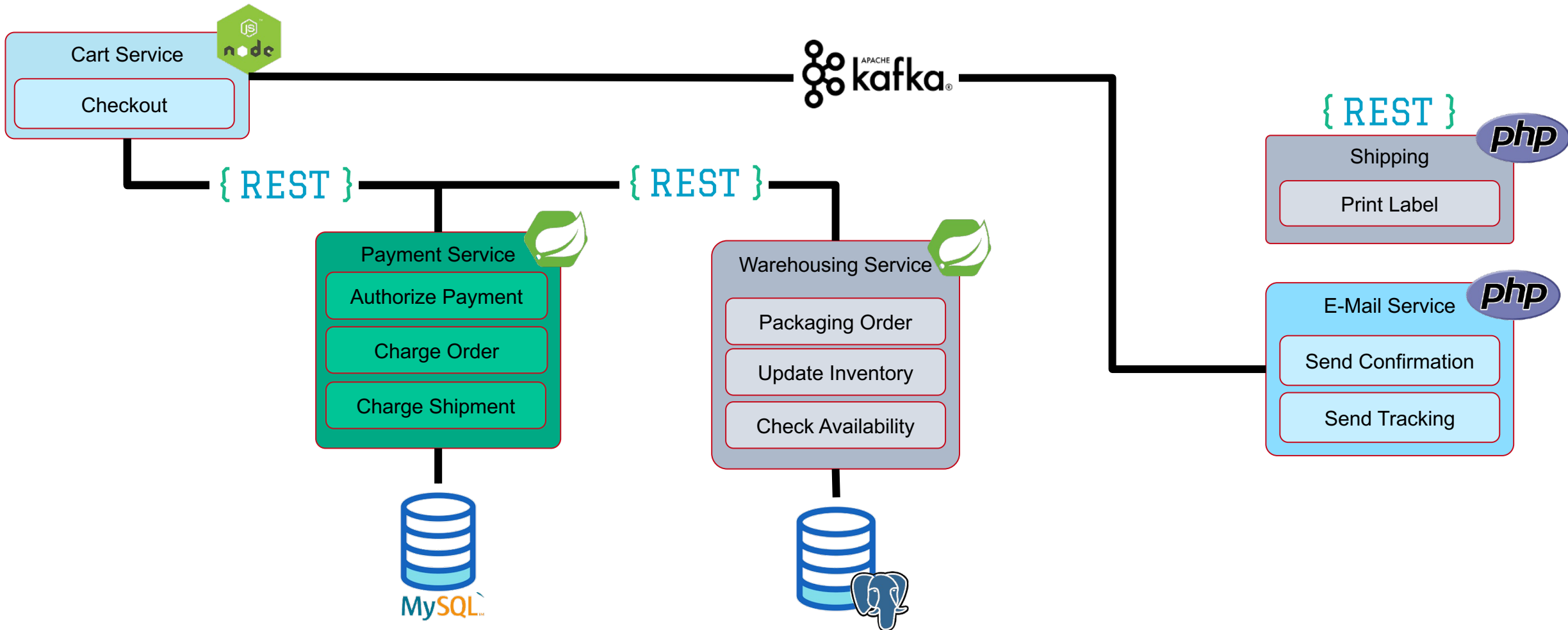
THE NEW STACK

Kubernetes - Overview

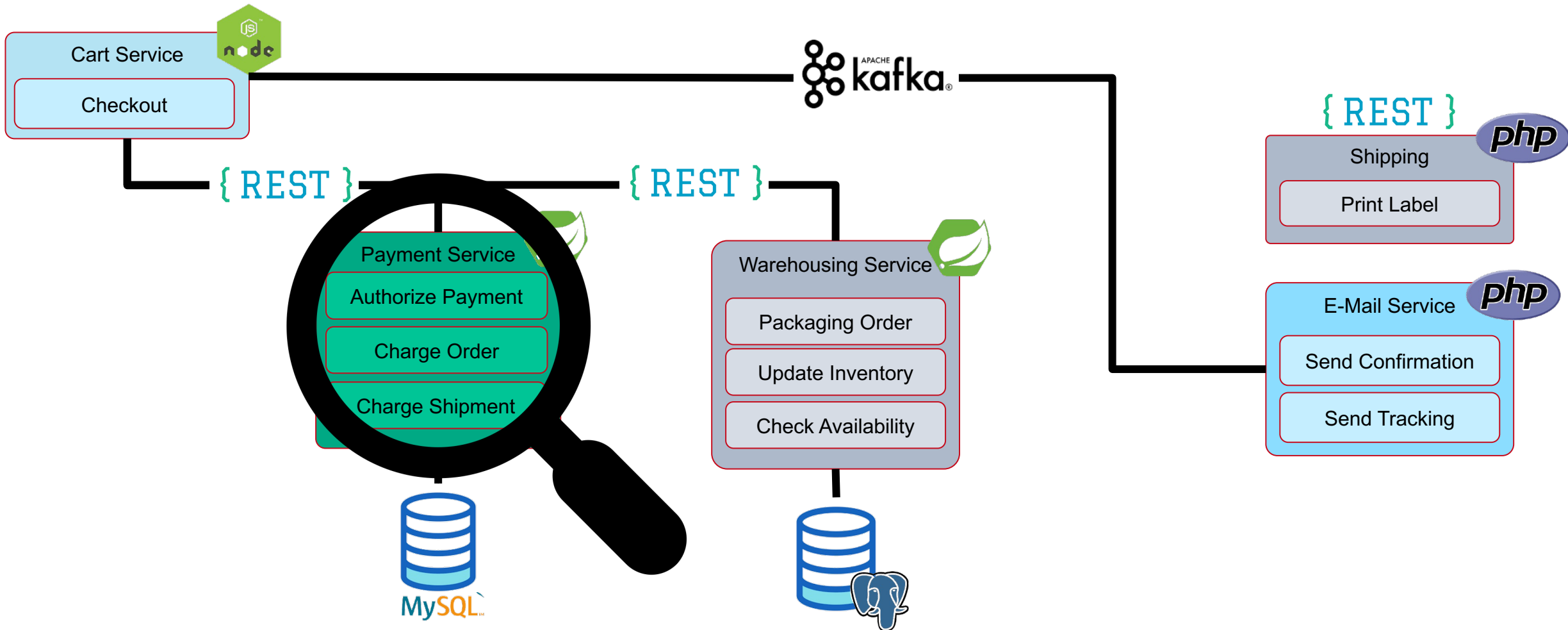


<https://medium.com/devops-mojo/kubernetes-architecture-overview-introduction-to-k8s-architecture-and-understanding-k8s-cluster-components-90e11eb34ccd>

Web shop example



Web shop example

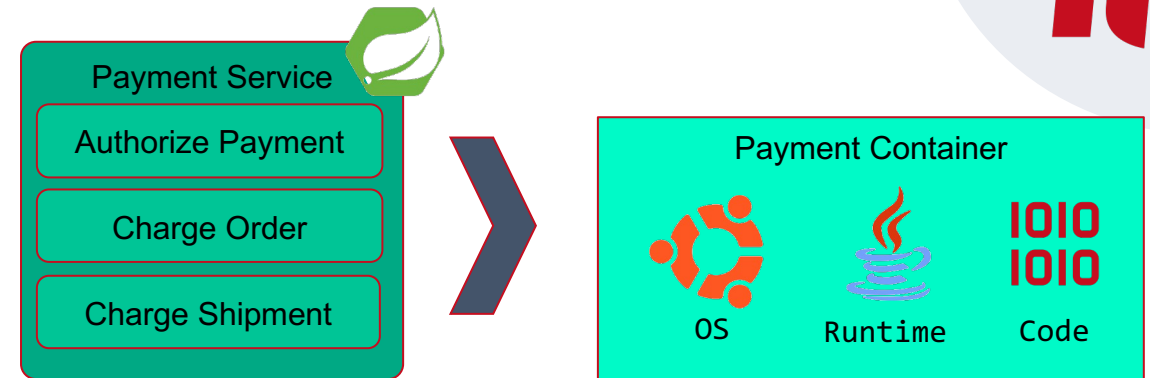


Deployment

- Each **Service** is packaged as a container
- We need to configure each Service as a **Deployment** of containers
- Databases and infrastructure service can:
 - run inside Kubernetes,
 - or be located externally

More than one container can be part of a service

Deployment Package:

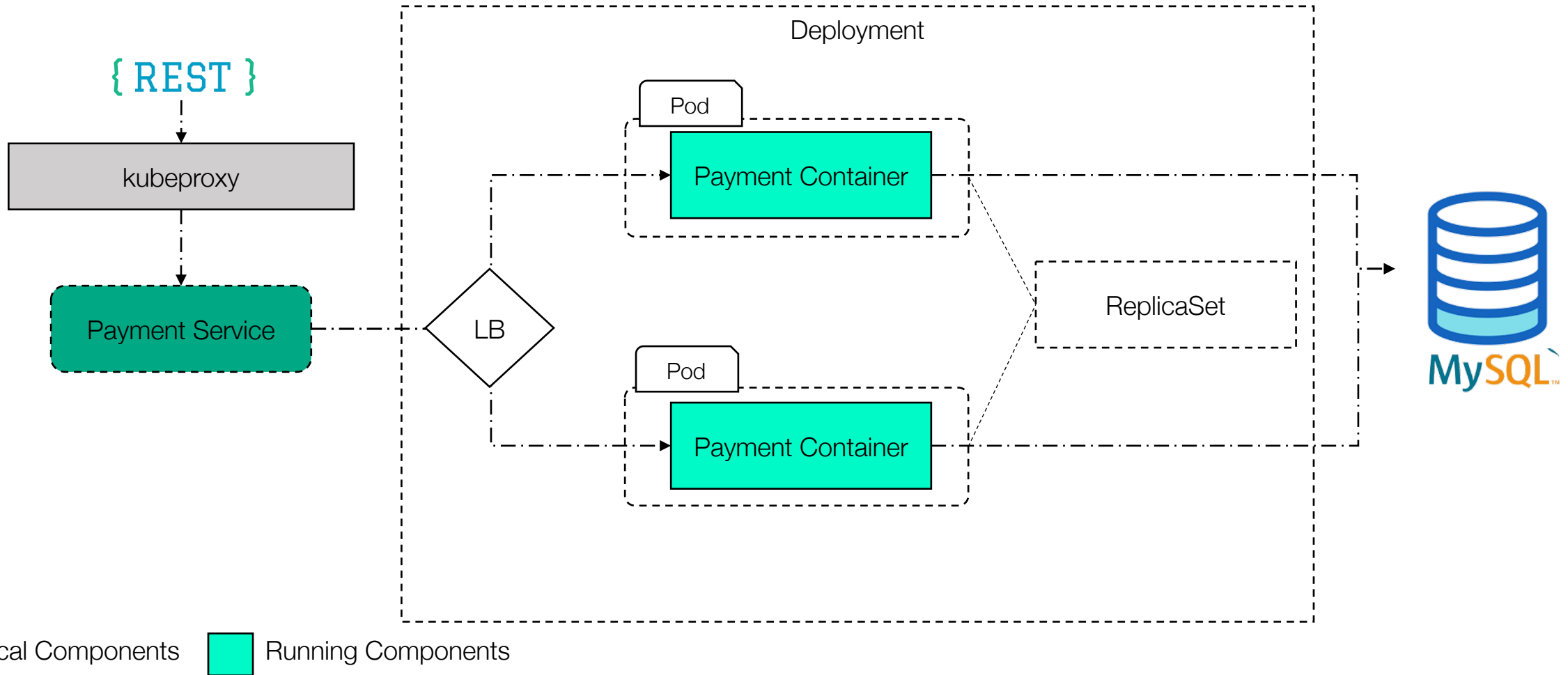


Deployment Configuration:

```
kind: Service
metadata:
  name: payment
spec:
  spec:
    ports:
      - port: 80
        protocol: TCP
    selector:
      run: payment-dep
  ----
kind: Deployment
metadata:
  name: payment-dep
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: payment-od
          image: payment:v1
```

payment.yaml

Deployment



Kubernetes Pods

Basic, **atomically deployable** unit in Kubernetes

A Pod consists of one or many co-located containers. A Pod represents a **single instance of an application**.

Each Pod has its own, uniquely assigned and **internal IP**

Pods are **mortal**, which means that if the node the Pod runs on becomes unavailable, the workload also goes unavailable.

my-first-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Slide adapted from: <https://speakerdeck.com/luxas/intro-to-the-cloud-native-world-of-kubernetes-january-2019?slide=28>

Kubernetes Deployment

- With a **Deployment**, you can manage Pods in a **declarative and upgradable manner**.
- **Replicas**: Kubernetes will make sure that the number of Pods created from the template always are available.
- When the Deployment is updated, Kubernetes will perform a **rolling update** of the Pods running in the cluster (i.e., creating one new Pod, and remove an old one, until all Pods are new; other deployment strategies are possible)

Slide adapted from: <https://speakerdeck.com/luxas/intro-to-the-cloud-native-world-of-kubernetes-january-2019?slide=29>

my-first-pod-and-deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
    name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
```

Pod template

```
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - image: nginx:1.13.9-alpine
        name: nginx
        ports:
          - name: http
            containerPort: 80
```

Kubernetes Service

- A **Service** exposes one or many Pods via a stable, immortal, and internal IP address
- It's also accessible via cluster-internal DNS,
`{service}.{namespace}.svc.cluster.local`
 Here: `nginx.default.svc.cluster.local`
- The **Service** selects Pods based on the label key-value **selectors** (here `app=nginx`).
- A **Service** may expose multiple ports.
- The **ClusterIP** can be declaratively specified, or dynamically allocated.

`my-first-service.yml`

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 80
      targetPort: 80
  selector:
    app: nginx
```

Slide adapted from: <https://speakerdeck.com/luxas/intro-to-the-cloud-native-world-of-kubernetes-january-2019?slide=30>

Kubernetes Ingress Controllers

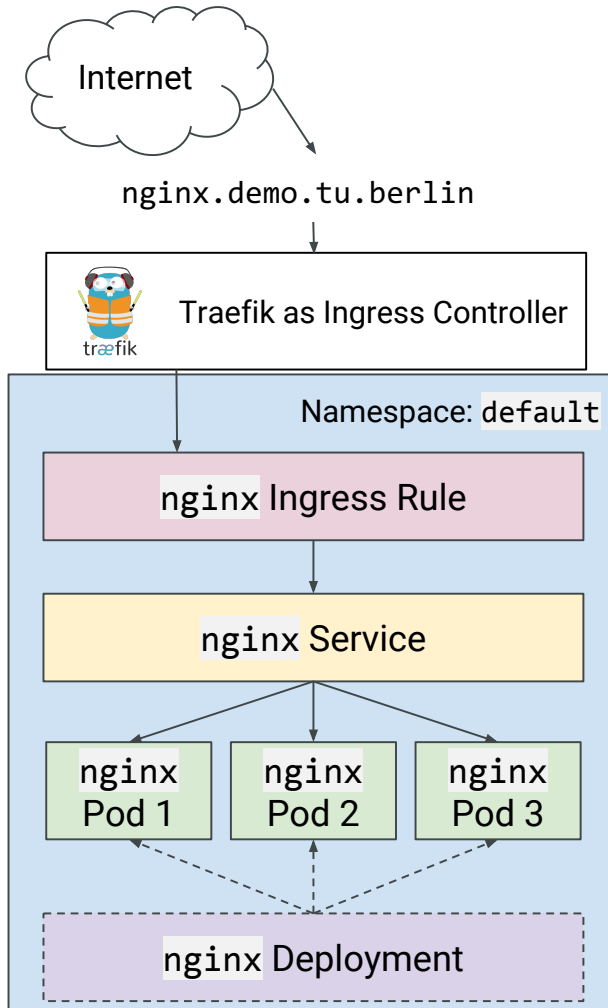
- A **Service** is only accessible inside of the cluster.
- To expose the Service to the internet, one must deploy an **Ingress** controller, like Traefik, and create an **Ingress Rule**.
- The Ingress controller is a load balancer that's creating forwarding rules based on the Ingress Rules in the Kubernetes API (see next slide)

my-first-ingress.yml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  rules:
    - host: nginx.demo.tu.berlin
  http: paths:
    - path: /
  backend:
    serviceName: nginx
    servicePort: 80
```

Slide adapted from: <https://speakerdeck.com/luxas/intro-to-the-cloud-native-world-of-kubernetes-january-2019>

Minimum working example

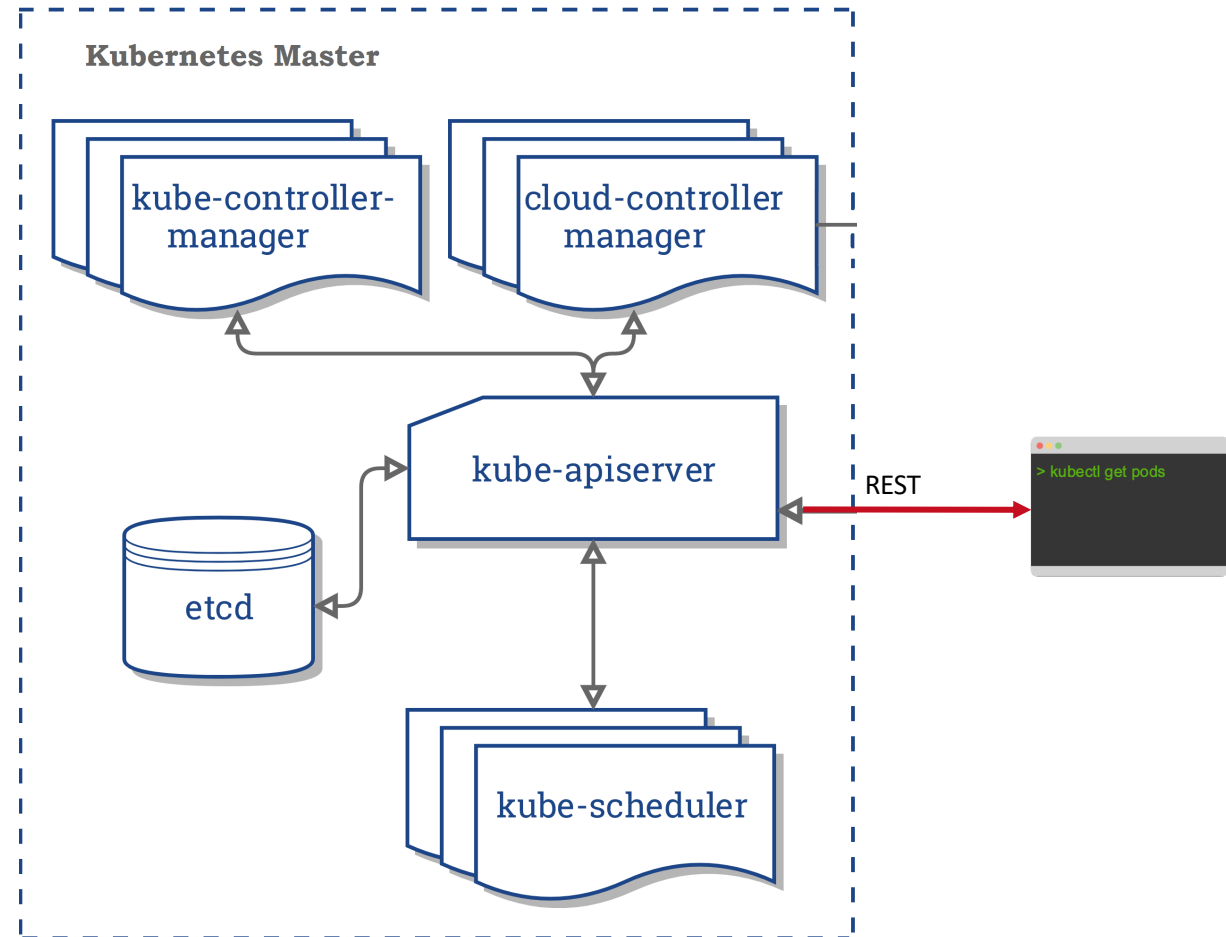


Note, your services will often be put into multiple namespaces for separating concerns.

Slide adapted from: <https://speakerdeck.com/luxas/intro-to-the-cloud-native-world-of-kubernetes-january-2019>

Kubernetes Control Plane

- Functionality of the *master components* is referred to as the control plane
- The control plane offers RESTful APIs for Kubernetes Object manipulation
- **kubectl** is the CLI for interacting with a Kubernetes cluster
 - kubectl is installed on your local machine
 - Interacts with RESTful API of Kubernetes



Pictures from:
<https://kubernetes.io/docs/concepts/overview/components/>
<https://dockerlabs.collabnix.com/kubernetes/beginners/what-is-kubect.html>

Setting up your first Kubernetes cluster

When you deploy an application to k8s, you define how many replicas of each service you'd like to run.

Scaling manually:

```
kubectl scale deployment payment-dep --replicas=4
```

Horizontal Autoscaler (deploys more pods):

```
kubectl autoscale deployment payment-dep --min=1 --max=4 --cpu-percent=50
```

Specify a maximum and minimum number of replicas and set a threshold for resources, e.g. CPU utilization

A new HorizontalPodAutoscaler object is created

Kube-Controller-Manager regularly queries resource utilization against the metrics specified in the HorizontalPodAutoscaler object and calculates the number of replicas accordingly

Can be combined with Elastic Autoscaler:

Deploys more Kubernetes nodes in response to high demand

Vertical Autoscaler (assigns more resources to a pod)

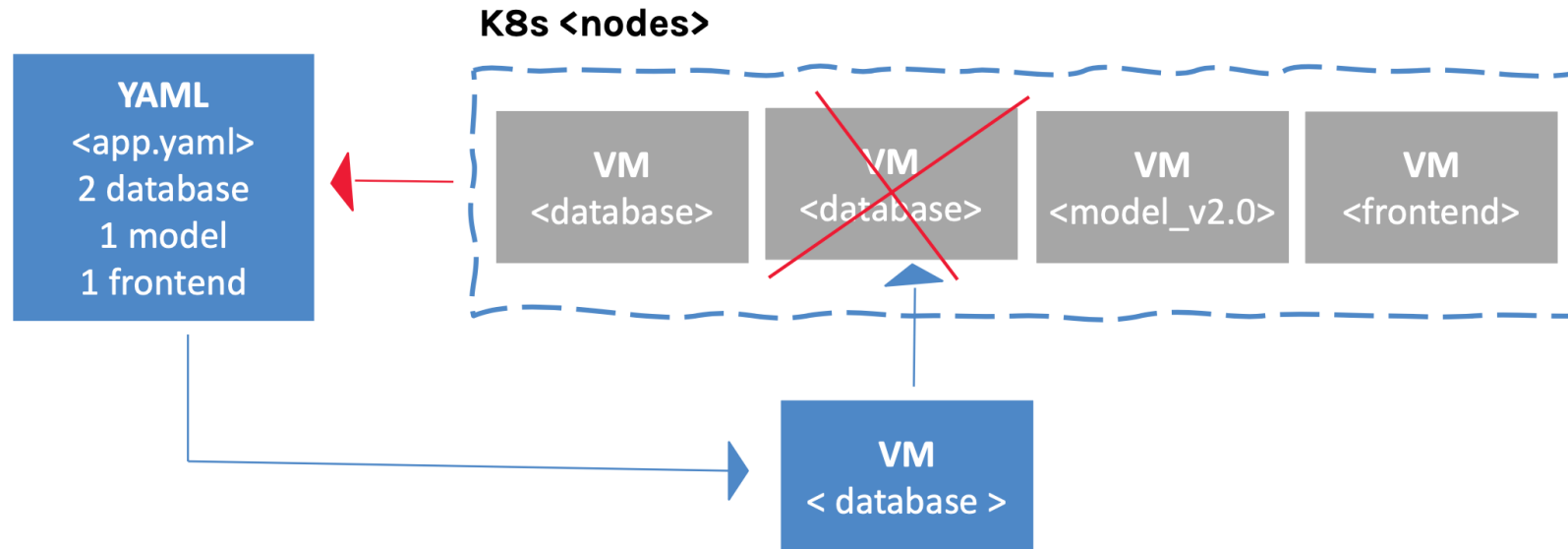
- The VerticalPodAutoscaler recommends or automatically adjusts values for CPU and memory

Deployment Configuration:

```
kind: Deployment
metadata:
  name: payment-dep
spec:
  replicas: 2
  template:
    spec:
      containers:
        - name: payment-od
          image: payment:v1
```

payment.yaml

Online self-healing infrastructure

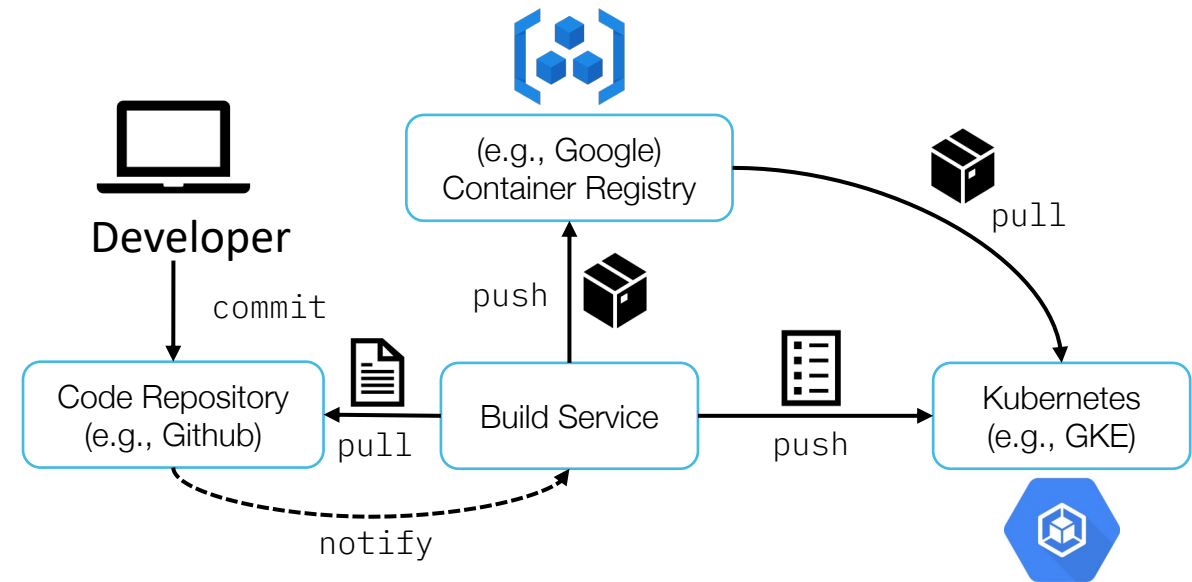


See also: Kubernetes Liveness, Readiness and Startup Probes

<https://harvard-iacs.github.io/2020-AC295/lectures/lecture3/presentation/lecture3.pdf>

Continuous Development

- Different degrees of automation and customization
- Insight into ongoing release processes
- Automated roll-back in case of failures
- Different roll-out strategies



Recommended tutorial for self-study:
<https://cloud.google.com/kubernetes-engine/docs/tutorials/gitops-cloud-build>

Try it yourself, e.g., with MiniKube

minikube

 build passing
 go report A+
 downloads 5.8M
 release v1.30.1



minikube implements a local Kubernetes cluster on macOS, Linux, and Windows. minikube's [primary goals](#) are to be the best tool for local Kubernetes application development and to support all Kubernetes features that fit.

```

time minikube start
minikube v1.13.0 on Darwin 10.15.6
Using the docker driver based on user configuration
Starting control plane node minikube in cluster minikube
Creating docker container (CPUs=2, Memory=3892MB) ...
Preparing Kubernetes v1.19.0 on Docker 19.03.8 ...
Verifying Kubernetes components...
Enabled addons: default-storageclass, storage-provisioner
kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" by default

Executed in 23.96 secs  fish           external
   usr time    1.66 secs  237.00 micros    1.66 secs
   sys time    0.78 secs   943.00 micros    0.78 secs

```

<https://github.com/kubernetes/minikube>

Common commands

```
kubectl create -f app-db-deployment.yaml
```

```
kubectl get deployment
```

```
kubectl get pods
```

```
kubectl get pods /  
-o=custom-columns=NAME:.metadata.name,IP:.status.podIP
```

```
kubectl create -f app-server-deploymnet.yaml
```

```
kubectl expose deployment / app-deployment --type=LoadBalancer --port=8080
```

```
kubectl get services
```

```
kubectl delete service app-deployment
```

```
kubectl delete deployment app-server-deployment
```

```
kubectl delete deployment app-db-deployment
```

<https://harvard-iacs.github.io/2020-AC295/lectures/lecture3/presentation/lecture3.pdf>

Recommended tooling

Helm charts, “a package manager for Kubernetes”

<https://helm.sh/>

Kubespray and others, Kubernetes installer

<https://github.com/kubernetes-sigs/kubespray>

Lens, exploring running clusters

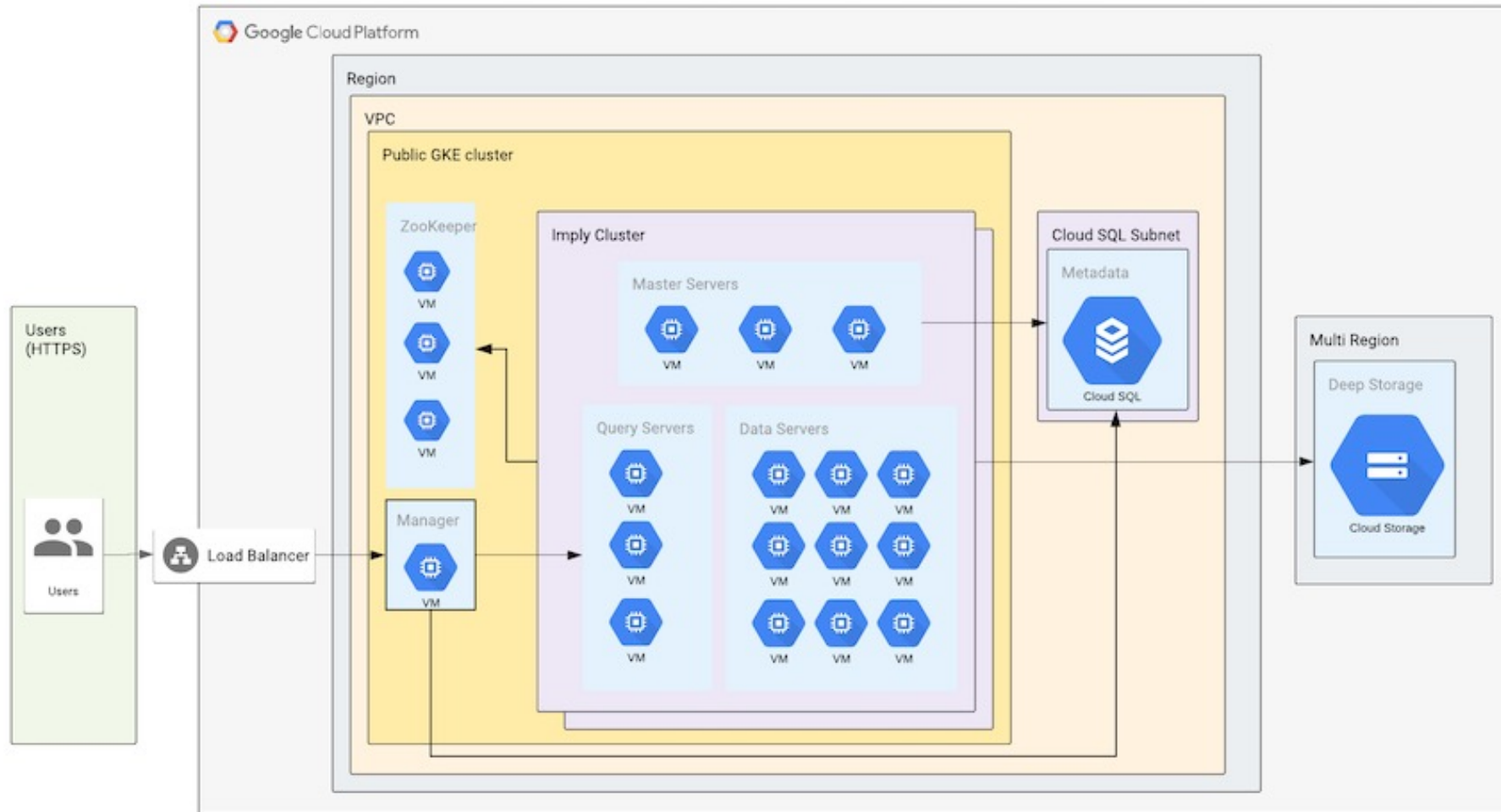
<https://k8slens.dev/>

<https://collabnix.github.io/kubetools/>



-  Cluster Management
-  Cluster with Core CLI tools
-  Alert and Monitoring
-  Logging and Tracing
-  Troubleshooting / Debugging
-  Development Tools/Kit
-  Alternative Tools for Development
-  CI/CD integration Tools
-  Security Tools
-  Network Policies
-  Testing Tools
-  Service Mesh
-  Observability
-  Machine Learning/Deep Learning
-  Compute Edge Tools
-  Kubernetes Tools for Specific Cloud
-  Storage Providers
-  Multiple Tools Repo
-  Cost Optimisation
-  Function as a Service FaaS
-  Artificial Intelligence

Managed Kubernetes, e.g., Google Kubernetes Engine (GKE) or Amazon Elastic Kubernetes Service (EKS)



<https://docs.implify.io/latest/k8s-gcp-enhanced/>

Some Challenges

Auto-scaling & perfect resource utilization

(e.g., AI Ops, multi-cluster, multi-cloud,...)

Observability & chaos engineering

(e.g., logging, tracing, monitoring)

Privacy (incl. security)

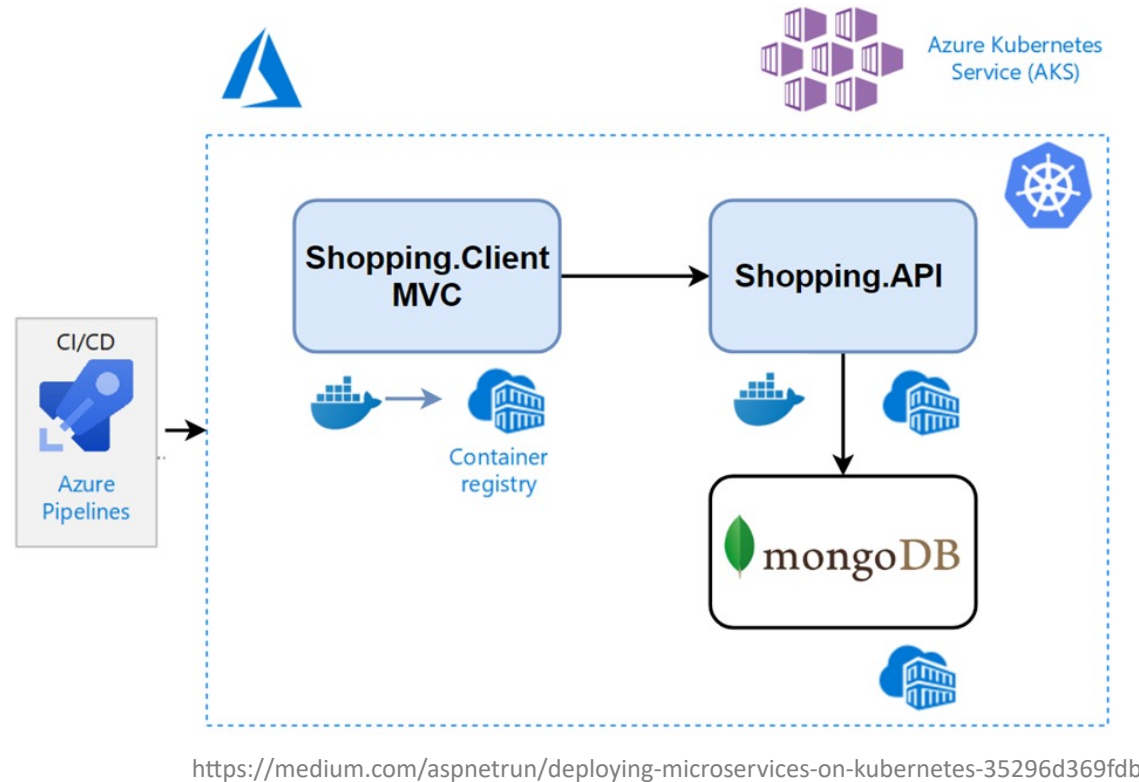
(e.g., Role-Based Access Control, network rules, firewalls, secret management etc.)

Infrastructure as Code, Release strategies, Service Discovery

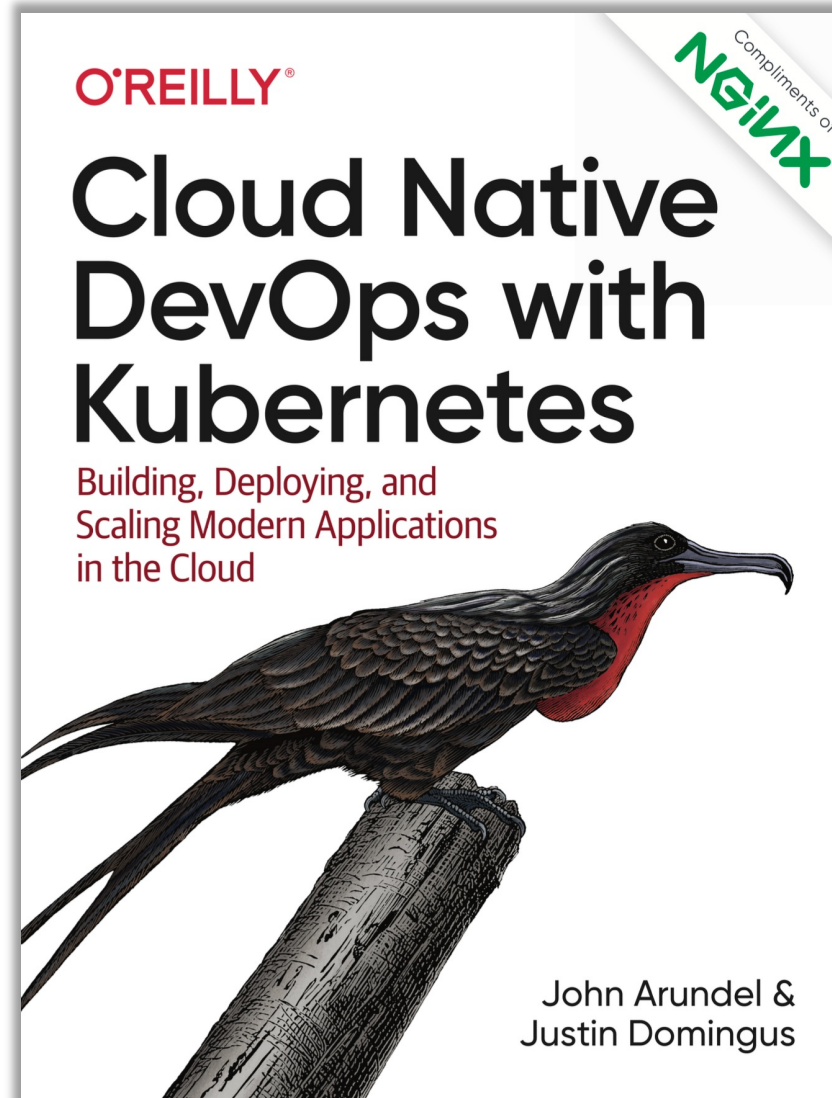


next lecture

Job Interview Question



Your external consultant proposes this architecture to you, how would you respond?



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Further reading

CDE tutorial: <https://circleci.com/blog/simplifying-your-ci-cd-build-pipeline-to-gke-with-circleci-orbs/>

- Uses Github, Docker, CircleCI and GKE



Kubernetes Documentation

<https://kubernetes.io/docs/concepts/overview/>

Extensive (>3.5h) video tutorial on Kubernetes

<https://www.youtube.com/watch?v=X48VuDVv0do>

Schedule (still preliminary)

